Evolution-Based Online Automated Machine Learning

Cedric Kulbach, Jacob Montiel, Maroua Bahri, Marco Heyden, Albert Bifet

Evolution-based Online Automated Machine Learning

Cedric Kulbach[1], Jacob Montiel[2], Maroua Bahri[3], Marco Heyden[1], and Albert Bifet[2]

[1] Research Center for Information Technology (FZI),
Haid-Und-Neu-Str. 10-14, 76131 Karlsruhe, Germany
kulbach@fzi.de & marco.heyden@kit.edu
[2] University of Waikato
Private Bag 3105, Hamilton 3240, New Zealand
abifet@waikato.ac.nz & jmontiel@waikato.ac.nz
[3] Inria Paris
2 Rue Simone IFF, 75012 Paris, France
maroua.bahri@inria.fr

Abstract. Automated Machine Learning (AutoML) deals with finding well-performing machine learning models and their corresponding configurations without the need of machine learning experts. However, if one assumes an online learning scenario, where an AutoML instance executes on evolving data streams, the question for the best model and its configuration with respect to occurring changes in the data distribution remains open. Algorithms developed for online learning settings rely on few and homogeneous models and do not consider data mining pipelines or the adaption of their configuration. We, therefore, introduce *EvoAutoML*, an evolution-based online learning framework consisting of heterogeneous and connectable models that supports large and diverse configuration spaces and adapts to the online learning scenario. We present experiments with an implementation of *EvoAutoML* on a diverse set of synthetic and real datasets, and show that our proposed approach outperforms state-of-the-art online algorithms as well as strong ensemble baselines in a traditional test-then-train evaluation.

Keywords: Incremental Learning · Ensemble Learning · Evolutionary Algorithm · Data Stream

1 Introduction

Automated Machine Learning (AutoML) has shown impressive performance on offline learning tasks in which the whole data are available at once. In contrast to stand-alone offline learning approaches, AutoML automates the data mining pipeline by concatenating different algorithms and applying hyperparameter optimization (HPO) techniques to find the best performing combination and configuration of models. The success of AutoML has lead to the development of multiple well-known frameworks, such as *autosklearn* [15, 16, 23], *TPOT* [26],

GAMA [18] or H_2O [34]. However, many real-world environments generate data continuously and indefinitely in the form of never-ending data streams [2, 17]. Unlike the batch setting, the unbounded nature of these data raises some practical and technical requirements that need to be addressed, where a stream algorithm [6]:

- **R1:** processes a single instance at a time,
- **R2:** processes each instance in a limited amount of time,
- **R3:** uses a limited amount of memory,
- **R4:** is ready to predict at any time,
- **R5:** is able to adapt to changes in the data distribution[4].

When retraining AutoML or other offline learning algorithms, a major part of these requirements is infringed. Data patterns may change in unforeseen ways leading to a decrease in the predictive performance of the machine learning model because the current learned model may be no more representative for the next upcoming data. As a result, offline learning AutoML algorithms might not recommend suitable models for future data without retraining the entire model (**R1**, **R2** infringed). In order to enable adaption to ever-evolving data streams current approaches; we either use (i) change detectors to decide if a model should be retrained [13, 24] or (ii) homogeneous ensemble learning techniques [31, 33]. Both approaches are not applicable in practice due to two main reasons: retraining AutoML algorithms is often computational expensive [13] (**R1**, **R4** infringed), especially in large search spaces. The second reason is that if large search spaces are acquired, pure ensemble techniques would lead to a large increase in the number of parallel trainings (**R3**, **R4** infringed).

Data streams evolve over time, just like natural environments change, so survival of the fittest, mutations, and offspring allow populations to adapt to such environmental changes. Evolutionary algorithms follow a similar concept, where by creating offspring and allowing for mutations, they mimic natural selection and let the fittest individuals move over to the next generation. In this manner, they enable the system to adapt to changing data patterns which makes them particularly well suited for Online Automated Machine Learning.

Our approach, *EvoAutoML*, takes up this idea and naturally adapts the population of algorithms and configurations if changes occur in the data. As a result, we are able to avoid expensive retraining of an AutoML learner and take advantage of ensemble learning techniques.

The proposed offline AutoML approaches are unable to work with evolving data streams because they allow, among others, several access to data instances and therefore broke the requirements of the streaming framework. Thus, for fair comparison, we evaluate our approach by comparing the relevant performance metrics on established datasets to related state-of-the-art online learning approaches through compliance with the defined requirements. The main contributions of this paper are summarized as follows:

[4] Changes in data distributions or patterns are also referred to as *concept drift* [36].

- We provide a formalization and implementation for adapting large algorithm and configuration search spaces to evolving data streams.
- We conduct a broad evaluation of the proposed approach against state-of-the-art algorithms.
- To foster reproducibility, the code and datasets employed in our work are available on GitHub[5]

2 Related Work

In this section, we present an overview of the related work for online AutoML. We first discuss relevant offline AutoML methods and then relevant (ensemble) algorithms for the Online Learning setting.

2.1 Automated Machine Learning

Generally, AutoML aims to automate a Machine Learning (ML) pipeline containing the steps of (i) data cleaning, (ii) feature engineering and (iii) algorithm modelling. It can be defined as the problem of automatically (without human intervention) producing test set predictions for a new dataset within a fixed computational budget [16]. To automate the data analysis pipeline, AutoML addresses the Combined Algorithm Selection and Hyperparameter (CASH) optimization problem [16]. The idea of AutoML was initially developed in [35], which combines the *WEKA* ML framework [21] with Bayesian optimization [12] to search for the best ML instance for a given dataset.

The most established frameworks for offline AutoML are *Auto-Weka 2.0* [25], *autosklearn* [15, 16, 23] , *TPOT* [26], *GAMA* [18], and H_2O [34], that mainly differ in their search space and HPO technique. As HPO technique, *Auto-Weka 2.0* exploits a random forest algorithm, *autosklearn* a Baysian optimization [23] approach, *TPOT* and *GAMA* employs evolutionary algorithms, and H_2O a grid search approach. Our approach, *EvoAutoML*, extends current batch AutoML approaches in order to make them applicable and suitable with evolving data streams.

2.2 Online Learning

Since data streams are potentially infinite and new observations may arrive with a high frequency, stream algorithms must be efficient in terms of resource usage, i.e. time **R3** and memory **R4** consumption. Fig. 1 exemplary shows how an online learning framework is able to comply with the stream requirements for a supervised learning task [6]. It processes each instance from an evolving data stream S, updates the underlying model, and is ready to predict at any time.

[5] https://github.com/kulbachcedric/EvOAutoML.git

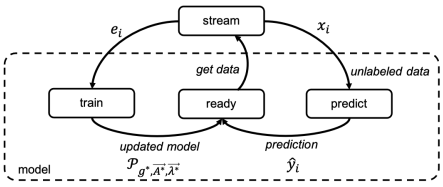

Fig. 1. Online Learning, following [28]

However, some algorithms have been specifically created and/or adapted to operate on data streams. For instances, *Hoeffding Tree* (*HT*) [14], *Hoeffding Adaptive Tree* (*HAT*) [5], *Logistic Regression*, ensemble methods such as *Online Bagging* (*OB*) [29]. Other algorithms require adaption so that they can be used in an online fashion, such as creating mini-batches or introducing a sliding window [3, 30]. If concept drift occurs in the online learning setting, the initially selected model may not longer be the optimal one. Ensemble techniques, such as *OB* with or without an ADWIN change detector [31], *Leveraging Bagging* (*LB*) [8] or *Adaptive Random Forest* (*ARF*) [19] have shown to be competent in adapting to temporal changes. *OB* [29, 30] propose an approach that updates a set of models by weighting each instance from the stream with a $Poisson(1)$ distributed number. Adding an ADWIN [4] change detector to *OB* enables dealing with concept drifts. *LB* [8] improves the *OB* approach by adding more randomization to the input and output of the classifier and therefore leverages the predictive performance. *ARF* [19], an adaption to the random forest algorithm [10], includes an effective resampling method that handles different types of concept drifts. Streaming Random Patches (SRP) [20] is also a ensemble method that combines random subspaces and bagging while using a strategy to detect drifts similar to the one introduced in ARF [19]. To adapt the configuration of heterogeneous algorithms to changing data streams, in [13], authors proposed an AutoML approach that uses different adaption strategies to retrain AutoML instances, such as H_2O, *Autosklearn* and *GAMA* [18], but without taking into account costly retrainings of offline AutoML instances. However, the presented ensemble approaches employ homogeneous algorithms with identical configurations. Assuming algorithm and hyperparameter search spaces, such as in *autosklearn* ($|\Lambda| = 110$ possible configurations), training base algorithms in the manner of ensembles (e.g., *OB* [31] and *LB* [8]) becomes increasingly inefficient and does not consider a combination of algorithms within the algorithm search space. Our approach therefore introduces an evolutionary adaption strategy that consider heterogeneous algorithms and configuration spaces to cope with different types of concept drifts.

3 Approach

The question of changing and adapting the configuration of an algorithm as well as the orchestration of models without infringing the requirements [6] for online learning remains open. Our online AutoML framework is inspired by the CASH problem, a Genetic Algorithm (GA) approach and extends *OB* [29, 30] to enable online training in a high-dimensional algorithm- and hyperparameter-search space. However, the CASH solution does not consider the adaption of parameters

in an evolving data stream environment so far, on the other hand, the established online ensemble algorithms are only capable of processing a small set of homogeneous algorithms. Whence, our proposal uses a GA approach which naturally adapts its configurations within a small ensemble (population) to enable the adaption of large algorithm- and hyperparameter-search spaces to evolving data streams.

3.1 Online CASH

We first define the online CASH problem to adapt to the online learning scenario. Following the definition from [37], a ML pipeline structure $g \in G$ can be modelled as an arbitrary directed acyclic graph (DAG), where each node represents an algorithm $A \in \mathcal{A}$.

Definition 1. *Online CASH, adapted from [16, 24]*
Let $\mathcal{A} = \{A^{(1)}, \ldots, A^{(R)}\}$ be a set of step independent algorithms, and let the hyperparameters of each algorithm $A^{(j)}$ have a domain $\Lambda^{(j)}$. Further, let $S = e_1, e_2, \ldots, e_t, \ldots$ be an ordered sequence of examples of possibly infinite length and let t be the current observed example. Further, let $S^- = e_0, \ldots, e_t$ be an ordered sequence of past examples. Each example $e_i = \{x_i, y_i\}$ is a tuple of p predictive attributes $x_i = (x_{i,1}, \ldots, x_{i,p})$ and the corresponding label y_i. Let $\mathcal{L}(\mathcal{P}_{g,\overrightarrow{A},\overrightarrow{\lambda}}(S^T), S^V)$ denote the loss that algorithm combination $P^{(j)}$ achieves on a subset of validation examples $S^V \subset S^-$ when trained on $S^T \subset S^-$ with hyperparameters $\overrightarrow{\lambda}$. Denote that $S^T \cap S^V = \emptyset$.

Then the Online CASH problem is to find the joint algorithm combination and hyperparmeter setting that minimizes the loss:

$$g^*, \overrightarrow{A}^*, \overrightarrow{\lambda}^* \in \underset{P^{(j)} \in \mathcal{P}, \lambda \in \Lambda^{(j)}, A \in \mathcal{A}, g \in G}{\arg\min} \mathcal{L}(\mathcal{P}_{g,\overrightarrow{A},\overrightarrow{\lambda}}(S^T), S^V) \tag{1}$$

Existing online (ensemble) algorithms do not fully cover the Online CASH problem. On the one hand, they do not consider a structure $g \in G$ and only cover a small range of hyperparameters $\lambda \in \Lambda$. On the other hand, their hyperparamters are usually set at the start of the stream and are not changed as the stream evolves. The range of covered hyper-parameters is restricted by the number of trained algorithms within the ensemble, whereas *OB* [29, 30], *ARF* [19], and other ensembles are based on homogeneous algorithms [27]. The graph structure g enables a combination and stacking of algorithms within $\mathcal{A}$ e.g. classifying a set of features x, after they have been scaled. Furthermore, Online CASH considers the configuration space Λ for each algorithm $A \in \mathcal{A}$.

Assuming large search spaces, such as those inherent in the number of existing algorithms and their configurations in the stream setting, a scaleable and adaptable approach becomes necessary. Therefore, by following Fig. 1, the introduced Def. 1, and the requirements defined in [6], we propose in Algo. 1 an AutoML training algorithm, that adapts to concept drifts in an online learning manner and is capable to handle large search spaces.

3.2 EvoAutoML

The core of our training and adaption procedure is a GA inspired by [32]. Algo. 1 shows the *EvoAutoML* algorithm. The input consists of a data stream S, a population size P, and a sampling rate f_{SS}. The length of the incoming stream S gives the number of updates $\lfloor t/f_{SS} \rfloor$ and is potentially infinite. Furthermore, our algorithm requires a sampling rate f_{SS}, which controls the rate at which a mutation is applied. Finally, we need a loss function $\mathcal{L}$ which estimates the performance of a pipeline configuration on given examples e_i (interleaved test-then-train evaluation) and a search space, containing all possible graph structures $g \in G$, algorithms $A \in \mathcal{A}$ and their configurations Λ. The algorithm is

Algorithm 1 EvoAutoML Training

1: **Input:**
2: Data stream S, population size P, sampling rate f_{SS}, loss function $\mathcal{L}$, configuration space $\mathcal{A}, \Lambda, G$
3: **Output:**
4: Set of suited algorithms configurations:
5: $p^* = \{\mathcal{P}^{(1)}, \ldots, \mathcal{P}^{(P)}\}$
6:
7: $p \leftarrow \emptyset$ ▷ Initialization
8: **while** $|p| < P$ **do**
9: $\mathcal{P} \leftarrow \text{Random}(G, \mathcal{A}, \Lambda)$
10: $p \leftarrow p \cup \mathcal{P}$
11: **end while**
12: $t \leftarrow 0$
13: **if** e_t **then** ▷ Start Datastream
14: **if** $t \mod f_{SS} == 0$ **then**
15: $\mathcal{P}^{best} \leftarrow \min_{\mathcal{P} \in p} \mathcal{L}(\mathcal{P}(S^T), S^V)$
16: $\mathcal{P}^{weak} \leftarrow \max_{\mathcal{P} \in p} \mathcal{L}(\mathcal{P}(S^T), S^V)$
17: $\mathcal{P}^{mut} \leftarrow \text{Mutate}(\mathcal{P}^{best})$
18: $p \leftarrow p \cup \mathcal{P}^{mut}$
19: $p \leftarrow p \backslash \mathcal{P}^{weak}$
20: **end if**
21: $\omega \leftarrow Poisson(6)$
22: **for** $\mathcal{P} \in p$ **do** ▷ Update Population
23: **loop** ω
24: $\mathcal{P}.\text{fit}(e_t)$
25: **end loop**
26: **end for**
27: $t \leftarrow t + 1$
28: **end if**

initialized (lines 7-12) by building a random population of algorithm pipelines $\mathcal{P}_{g, \vec{A}, \vec{\lambda}}$. Notice that by initializing p with random online learning pipelines, the algorithm is able to predict at any time (**R4**).

In line 13, the data stream starts and a mutation is applied with a rate of f_{SS} (lines 14-20). Within the mutation steps the algorithm selects, in the first step, the best $\mathcal{P}^{best}$ and weakest $\mathcal{P}^{weak}$ pipeline configuration. Based on the best pipeline $\mathcal{P}^{best}$ configuration the mutation is applied in line 17, where similar to [32] a random parameter of $\mathcal{P}^{best}$ is changed within $\mathcal{A}$ and Λ, passed to $\mathcal{P}^{mut}$ and added to p. The weakest pipeline $\mathcal{P}^{weak}$ is removed from p (line 19). After the mutation step, the population is trained on the new instance e_i (lines 21-25)

similar to *OB* [30] with a $\omega \sim Poisson(6)$ distribution. The choice for the distribution results from *LB* [8].

Our approach respects the requirements of [6] (**R1**-**R5**) by processing each example e_i at a time (**R1**) and in a limited amount of time (**R2**), e.g. by adjusting the population size P (see also Sec. 4). Since our approach has access to a population of trained pipelines at each point in time of the data stream *EvoAutoML* is also able to predict at any time (**R4**). Algo. 1 updates the population in an ensemble manner within the training process to search for suited configurations that can also be used for prediction. To predict for an unlabelled instance (see Fig. 1), our approach uses a hard majority voting approach of the algorithm configurations in p to predict the label $\hat{y}_i = mode\{\mathcal{P}.predict(e_i) \in p\}$.

In contrast to existing techniques which only consider the problem of algorithm selection, our approach also takes into account the configuration space Λ from a a range of algorithms $\mathcal{A}$ and the pipeline structure g. As a result, *EvoAutoML* addresses the complete Online CASH problem while other approaches can only deliver partial solutions.

4 Experiments

In this section, we describe our evaluation, present the baseline algorithms, introduce the datasets used for evaluation, and discuss the experimental setup. To evaluate our approach, we apply the interleaved test-then-train evaluation, which is a commonly used approach in data stream settings. Here, each incoming instance first serves for testing the current performance of the algorithm and afterwards for training and updating the algorithm. In addition to the fulfillment of the requirements **R1** and **R4** (see Sec.1), we show that our approach is able to outperform related algorithms by evaluating (i) the final accuracy (**R5**), (ii) the avg. time required (**R2**) to process selected datasets, and (iii) the memory consumption (**R3**). We show that *EvoAutoML* is compatible with recent online algorithms and thus fulfills the requirements of [6] (**R1**-**R5**). In Tab. 1, we present the stream datasets as well as the synthetic data stream generators used within the evaluation.

Table 1. Datasets

Name	Variables	#Samples	#Features	#Classes
RBF(a,b) [7]	a: #centroids b: moving speed	$1M$	50	5
SEA(a) [23]	a: changing width	$1M$	3	2
Agrawal(a) [1]	a: changing width	$1M$	9	2
LED() [11]		$1M$	24	7
HYP(a,b) [22]	a: #features b: magnitude change	$1M$	50	2
SINE() [17]		$1M$	2	2
Covertype [7]		581,012	54	7
Elec [7]		45,312	6	2

4.1 Search Space

Our approach uses two algorithm types that can be categorised into (i) *preprocessors* $A_{(i)}$ and (ii) *predictors* $A_{(ii)}$, and can be variably linked with each other. The preprocessing step can either be a *missing value cleaner*, *min-max scaler*, or a *standard scaler* ($|A_{(i)}| = 3$). The prediction step contains *Gaussian Naive Bayes* (*GNB*), *HT*, *k-Nearest Neighbors* (*KNN*), and *Logistic Regression* classifiers. In total, the classification step contains $|A_{(ii)}| = 4$ and therefore $3 \times 4 = 12$ possible algorithm configurations.

All algorithms $A^{(i)}$ can be parametrized by their domain $\Lambda^{(i)}$. For example, the *KNN* classifier can be parameterized by the number of neighbors, or the *HT* classifier by its maximal depth or the tie threshold as well as by the binary parameters if a binary split strategy should be applied or if poor attributes should be removed. On the whole, our domain space contains 174 possible pipeline configurations. Here, the advantage of our approach (Algo. 1) comes apparent. While current ensemble and boosting methods are based on homogeneous models (pipelines $\mathcal{P}$), *EvoAutoML* is capable of handling a diverse set of pipelines and pipeline configurations $G, \mathcal{A}$.

4.2 Experimental Setup

We implemented *EvoAutoML* on top of *River* [27], the source code is made publicly available[6]. We evaluated our approach with a population size $P = 10$ and a sampling rate $f_{SS} = 1000$. The population size is chosen equal to the size of the ensemble learners. Furthermore, the choice for the population size and rate is two-folded: First, all algorithms within the population are trained in an ensemble and thus a high population size or sampling rate would lead to computational expensive training updates. Second, during the implementation, the configuration $P = 10$, $f_{SS} = 1000$ was found to be a compromise between predictive performance and the amount of resources required. To compare our approach with established ensemble learners, one can set (i) an equal algorithm space $\mathcal{A}$ to all ensemble learners or (ii) compare our approach to the preset configurations. However, by setting an equal algorithm space $\mathcal{A}$, with or without consideration of further configuration Λ, we pursue the question of the search for the best performing parameterization, that ensemble learners answer by training all algorithms in $\mathcal{A}$ in a parallel manner. In contrast, by using the default configuration of each ensemble learner, we pursue the question for the best performing approach. Regarding the computational complexity for large search spaces, we evaluated the related algorithms in their proposed configuration to pursue the question for the best performing approach.

To cover a broad range and the most suitable incremental algorithms, we evaluate *EvoAutoML* against *HT* [14], Gaussian NB, and *KNN* classifiers. Since *EvoAutoML* contains a population of pipelines $P_{g,\vec{A},\vec{\lambda}}$, we also evaluate our approach against ensemble learners such as *ARF* [8], *LB* [8] and *OB* [19, 31]

[6] https://github.com/kulbachcedric/EvOAutoML.git

using *HT* as base classifier. Each ensemble learner contains 10 base classifiers. Each instance from the stream is transformed using a standard scaler (zero mean and unit variance) before passing it to the ML algorithm. All baseline algorithms are carried out with their default configuration.

To conclude our experimental setup, we evaluated the predictive performance, the total running time, and memory consumption of our approach with the search space proposed in Sec. 4.1 against the proposed baseline algorithms.

5 Results

In this section, we present the results of our approach and show that our approach is able to adapt to temporal changes, outperforms the state-of-the-art algorithms in predictive performance and has comparable computational costs as other online ensemble approaches. To show the adaption to temporal changes by following the requirements of [6] (esp. **R5**), we exemplary depict, in Fig. 2, the learning curves of our approach and the baseline approaches presented in 4.2 for the Covertype dataset.

The Covertype dataset [7] contains labeled instances of forest cover type (7 classes) from the US Forest Service, where 581,012 samples (measured in 30 x 30 meter cells) are characterized by 54 attributes. It has been used in several papers on data stream classification [9] and shows exemplary the adaptanility of *EvoAutoML* against other streaming classifiers (see Sec. 4.2). One can see in Fig 2 that *EvoAutoML* adapts to changes faster than *LB*, *ARF* and *OB*. While the other approaches show decreases in their accuracy at $\sim 250,000$ evaluated instances, *EvoAutoML* remains stable. Furthermore, we compare the training and testing time incurred by each model. The graph is in agreement with the statements from [19], and shows that single algorithms (*HT*, *GNB* and *KNN*) have the the lowest running time. For the ensemble learners, *ARF* has the lowest running time, followed by *LB*, *EvoAutoML* and *OB*. The evaluation of *EvoAutoML* takes slightly more time than *LB*, whereby the running time of *OB* increases faster than *LB* and *EvoAutoML*. In addition to other ensemble learners, *EvoAutoML* is able to adapt its parameters during the datastream based on a given loss function $\mathcal{L}$. This allows the adaptation to the data stream in terms of accuracy, but could also incor-

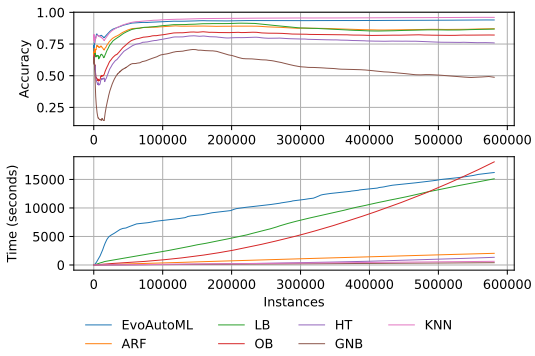

Fig. 2. Accuracy curve and time (in seconds) for *EvoAutoML*and baseline algorithms

porate metrics such as latency or memory consumption. However, Tab. 2 and Tab. 3 already show that *EvoAutoML* is competitive in terms of the final percentage of correctly classified examples, memory and the time consumption with an underlying accuracy loss. Tab. 2 compares *EvoAutoML* against the baseline

Table 2. Accuracy comparison of *EvoAutoML* against baselines. Accuracy is measured as the final percentage of examples correctly classified. The best individual accuracies are indicated in boldface

Dataset	EvoAutoML	HT	GNB	KNN	ARF	LB	OB
Agrawal(50)	99.02 ±0.01	98.09 ±0.01	62.31 ±0.09	55.73 ±0.02	94.98 ±0.95	**99.69** ±0.00	98.46 ±0.01
Agrawal(50000)	94.43 ±0.02	91.84 ±0.02	62.33 ±0.09	55.52 ±0.02	93.03 ±0.93	**97.52** ±0.01	92.89 ±0.02
HYP(50,0.0001)	**87.51** ±0.02	84.38 ±0.00	91.61 ±0.01	67.93 ±0.00	71.19 ±0.71	84.54 ±0.01	87.14 ±0.01
HYP(50,0.001)	83.69 ±0.01	81.79 ±0.01	80.83 ±0.02	68.01 ±0.00	71.67 ±0.72	**83.95** ±0.01	84.43 ±0.01
LED()	**76.49** ±0.01	75.95 ±0.01	76.48 ±0.01	66.6 ±0.00	76.47 ±0.76	76.48 ±0.01	76.42 ±0.01
RBF(10,0.0001)	99.82 ±0.00	89.32 ±0.03	65.86 ±0.09	**100** ±0.00	99.85 ±0.01	99.64 ±0.00	98.07 ±0.00
RBF(10,0.001)	99.63 ±0.00	77.61 ±0.02	39.75 ±0.11	**99.99** ±0.00	99.22 ±0.99	99.01 ±0.00	93.68 ±0.01
RBF(50,0.0001)	97.51 ±0.01	83.05 ±0.03	35.26 ±0.13	**99.83** ±0.00	98.21 ±0.98	98.71 ±0.01	96.17 ±0.01
RBF(50,0.001)	96.99 ±0.01	48.15 ±0.04	25.32 ±0.07	**99.80** ±0.00	94.31 ±0.94	93.56 ±0.01	71.87 ±0.03
SINE()	**99.87** ±0.00	99.63 ±0.01	93.62 ±0.00	98.75 ±0.00	99.74 ±0.01	99.68 ±0.00	99.77 ±0.01
SEA(50)	98.99 ±0.00	97.78 ±0.01	95.65 ±0.00	97.23 ±0.00	99.64 ±0.01	**99.67** ±0.01	98.34 ±0.01
Elec	**88.09** ±0.01	79.61 ±0.02	72.87 ±0.03	79.53 ±0.01	87.79 ±0.88	87.32 ±0.01	81.74 ±0.02
Covertype	**91.09** ±0.07	66.67 ±0.10	63.64 ±0.11	73.74 ±0.12	89.7 ±0.09	90.41 ±0.08	83.66 ±0.12
Avg. Acc.	**93.32**	82.61	66.58	81.74	90.45	93.09	89.43
Avg. Rank	2.08	5.31	5.92	4.77	3.46	2.54	3.85

approaches presented in Sec. 4.2. It shows that *EvoAutoML* outperforms the baseline algorithms with an average final avg. accuracy of 93.32%. Comparing *EvoAutoML* against the single best algorithms, *EvoAutoML* performs 10.71% better on average. However, beside the strong results of our approach and the chosen ensemble learners, 2 also shows, that in the case of *KNN* classifiers on the RBF dataset, single best algorithms might perform marginally better than the ensemble learners. This slightly better performance on the RBF dataset may be the result of (i) an unsuitable baseline algorithm for the ensemble learners, or the transition gap to a suitable pipeline in the case of *EvoAutoML*. Comparing our approach against the ensemble algorithms, *EvoAutoML* slightly outperforms them with 0.32% on average. Taking the average rank into account *EvoAutoML* performs best with an avg. rank of 2.08. Furthermore, all ensemble algorithms perform better on the avg. accuracy and the avg. rank than the chosen single algorithms. Tab. 3 records the memory consumption, as well as the used RAM-hours and the avg. time consumption. One RAM-Hour equals to 1 Gb of RAM deployed for 1 hour and is accumulated over the generators and datasets. It shows that the significantly better performance of the ensemble learners is accompanied by higher memory and time consumption than with single algorithms. However, comparing the deployed RAM-hours and the time consumption of the ensemble learners, our approach consumes a fraction of the memory in terms of deployed RAM-hours and manages to iterate the quickest over the data stream.

In summary, we show beside the requirements **R1** and **R4** (see Sec. 3) that *EvoAutoML* meets the requirements **R2** and **R3** of [6] by consuming less time and memory as state-of-the-art ensemble learners. *EvoAutoML* outperforms these ensemble learners in a common test-then-train evaluation, which shows the ability to adapt (**R5)** to changes in the data distribution.

Table 3. Comparison of memory consumption (in MB) and Avg. Time (in s). One RAM-Hour equals to 1 Gb of RAM deployed for 1 hour.

Dataset	**EvoAutoML**	**HT**	**GNB**	**KNN**	**ARF**	**LB**	**OB**
Agrawal(50)	17.609	0.604	0.013	0.455	11.093	12.205	6.008
Agrawal(50000)	56.854	2.223	0.013	0.455	12.920	37.600	21.501
HYP(50,0.0001)	104.576	18.287	0.066	2.020	229.900	528.847	180.870
HYP(50,0.001)	127.877	18.516	0.066	2.020	356.600	395.203	187.146
LED()	35.954	2.104	0.048	0.379	10.133	39.723	18.570
RBF(10,0.0001)	24.527	13.359	0.133	2.020	25.803	22.897	134.988
RBF(10,0.001)	36.107	30.530	0.133	2.020	11.668	4.893	291.346
RBF(50,0.0001)	64.458	24.165	0.166	2.020	27.117	35.643	236.124
RBF(50,0.001)	29.288	9.173	0.166	2.020	25.453	8.023	98.340
SINE()	9.760	0.421	0.004	0.169	14.622	11.128	4.211
SEA(50)	17.833	0.716	0.005	0.205	8.408	14.070	7.454
Elec	12.697	0.205	0.012	0.417	6.850	1.729	1.938
Covertype	12.082	0.125	0.080	2.170	4.750	15.549	19.368
Avg. Time	33,638	4,635	1,489	2,119	56,786	58,347	35,243
RAM-Hours	7.19	0.32	0	0.01	50.38	44.55	24.35

6 Conclusion

In this paper, we propose an approach for evolution-based online automated machine learning that extends the CASH problem to the stream setting and adapts the hyperparameter search to work with data streams. The adaption of hyperparameters and the possibility of algorithm pipelines, showed that an evolutionary approach is able to outperform state-of-the-art single and ensemble-based methods. We evaluated *EvoAutoML* on performance metrics, as well as the total running time and the used memory as efficiency metrics on several common online learning generators and datasets.

References

1. Agrawal, R., Imielinski, T., Swami, A.N.: Database mining: A performance perspective. IEEE TKDE **5**(6), 914–925 (1993)
2. Alberg, D., Last, M., Kandel, A.: Knowledge discovery in data streams with regression tree methods. Wiley Interdiscip DMKD **2**(1), 69–78 (2012)
3. Bahri, M., Bifet, A., Gama, J., Gomes, H.M., Maniu, S.: Data stream analysis: Foundations, major tasks and tools. Wiley Interdiscip: DMKD **11**(3), e1405 (2021)
4. Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing. In: SIAM ICD. pp. 443–448 (2007)
5. Bifet, A., Gavaldà, R.: Adaptive learning from evolving data streams. In: Adams, N.M., Robardet, C., Siebes, A. (eds.) IDA. pp. 249–260. Springer (2009)
6. Bifet, A., Gavaldà, R., Holmes, G., Pfahringer, B.: Machine learning for data streams: with practical examples in MOA. MIT Press (2018)
7. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: massive online analysis. JMLR **11**, 1601–1604 (2010)
8. Bifet, A., Holmes, G., Pfahringer, B.: Leveraging bagging for evolving data streams. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD. vol. 6321, pp. 135–150. Springer (2010)
9. Bifet, A., Read, J., Zliobaite, I., Pfahringer, B., Holmes, G.: Pitfalls in benchmarking data stream classification and how to avoid them. In: Blockeel, H., Kersting, K., et al. (eds.) ECML PKDD. vol. 8188, pp. 465–479. Springer (2013)
10. Breiman, L.: Random forests. ML **45**(1), 5–32 (2001)

11. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth (1984)
12. Brochu, E., Cora, V.M., de Freitas, N.: A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. CoRR (2010)
13. Celik, B., Vanschoren, J.: Adaptation strategies for automated machine learning on evolving data. CoRR **abs/2006.06480** (2020)
14. Domingos, P.M., Hulten, G.: Mining high-speed data streams. In: Ramakrishnan, R., Stolfo, S.J., Bayardo, R.J., Parsa, I. (eds.) SIGKDD. pp. 71–80. ACM (2000)
15. Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., Hutter, F.: Auto-Sklearn 2.0: The Next Generation. CoRR (2020)
16. Feurer, M., Klein, A., Eggensperger, e.a.: Efficient and Robust Automated Machine Learning. In: Cortes, C., Lawrence, N.D., Lee, D.D. (eds.) Advances in Neural Information Processing Systems 28: NIPS. pp. 2962–2970 (2015)
17. Gama, J., Medas, P., Castillo, G., Rodrigues, P.P.: Learning with drift detection. In: Bazzan, A.L.C., Labidi, S. (eds.) SBIA. vol. 3171, pp. 286–295 (2004)
18. Gijsbers, P., Vanschoren, J.: GAMA: genetic automated machine learning assistant. J. Open Source Softw. **4**(33), 1132 (2019)
19. Gomes, H.M., Bifet, A., Read, J., Barddal, J.P., Enembreck, F., Pfahringer, B., Holmes, G., Abdessalem, T.: Adaptive random forests for evolving data stream classification. ML **106**(9-10), 1469–1495 (2017)
20. Gomes, H.M., Read, J., Bifet, A.: Streaming random patches for evolving data stream classification. In: ICDM. IEEE (2019)
21. Hall, M.A., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P.: The WEKA data mining software: an update. SIGKDD Explor. **11**(1), 10–18 (2009)
22. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: SIGKDD. pp. 97–106. ACM (2001)
23. Hutter, F., Kotthoff, L., Vanschoren, J. (eds.): Automated Machine Learning - Methods, Systems, Challenges. Springer (2019)
24. Imbrea, A.: An empirical comparison of automated machine learning techniques for data streams. B.S. thesis, University of Twente (2020)
25. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. JMLR **18**, 25:1–25:5 (2017)
26. Le, T.T., Fu, W., Moore, J.H.: Scaling tree-based automated machine learning to biomedical big data with a feature set selector. Bioinformatics **36**(1) (2020)
27. Montiel, J., Halford, M., Mastelini, S.M., Bolmier, G., Sourty, R., Vaysse, R., Zouitine, A., Gomes, H.M., Read, J., Abdessalem, T., Bifet, A.: River: machine learning for streaming data in python (2020)
28. Montiel, J., Read, J., Bifet, A., Abdessalem, T.: Scikit-multiflow: A multi-output streaming framework. JMLR **19**, 72:1–72:5 (2018)
29. Oza, N.C.: Online bagging and boosting. In: ICSMC. pp. 2340–2345. IEEE (2005)
30. Oza, N.C., Russell, S.J.: Experimental comparisons of online and batch versions of bagging and boosting. In: Lee, D., Schkolnick, M., Provost, F.J., Srikant, R. (eds.) ACM SIGKDD. pp. 359–364. ACM (2001)
31. Oza, N.C., Russell, S.J.: Online bagging and boosting. In: Richardson, T.S., Jaakkola, T.S. (eds.) Workshop on AISTATS (2001)
32. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: AAAI. pp. 4780–4789. AAAI (2019)

33. van Rijn, J.N., Holmes, G., Pfahringer, B., Vanschoren, J.: Having a blast: Meta-learning and heterogeneous ensembles for data streams. In: Aggarwal, C.C., Zhou, Z., Tuzhilin, A., Xiong, H., Wu, X. (eds.) ICDM. pp. 1003–1008 (2015)
34. Stetsenko, P.: Machine Learning with Python and H2O (2020), http://docs.h2o.ai/h2o/latest-stable/h2o-docs/booklets/PythonBooklet.pdf
35. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: SIGKDD. pp. 847–855. ACM (2013)
36. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. ML **23**(1), 69–101 (1996)
37. Zöller, M., Huber, M.F.: Survey on automated machine learning. CoRR **abs/1904.12054** (2019)